Available online at www.sciencedirect.com
SCIENCE DIRECT
Journal of Discrete Algorithms 2 (2004) 471–500
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

Approximating schedules for dynamic process graphs efficiently

Andreas Jakoby [1], Maciej Liśkiewicz [*,2], Rüdiger Reischuk

Institut für Theoretische Informatik, Universität zu Lübeck, Razeburger Allee 160, 23538 Lübeck, Germany

Available online 12 May 2004

Abstract

A model for parallel and distributed programs, the *dynamic process graph* (DPG), is investigated under graph-theoretic and complexity aspects. Such graphs embed constructors for parallel programs, synchronization mechanisms as well as conditional branches. They are capable of representing all possible executions of a parallel or distributed program in a very compact way. The size of this representation can be as small as logarithmic with respect to the size of any execution of the program.

In a preceding paper [A. Jakoby, et al., Scheduling dynamic graphs, in: Proc. 16th Symposium on Theoretical Aspects in Computer Science STACS'99, LNCS, vol. 1563, Springer, 1999, pp. 383–392] we have analysed the expressive power of the general model and various variants of it. We have considered the scheduling problem for DPGs given enough parallelism taking into account communication delays between processors when exchanging data. Given a DPG the question arises whether it can be executed (that means whether the corresponding parallel program has been specified correctly), and what is its minimum schedule length.

In this paper we study a subclass of dynamic process graphs called PAR-output DPGs, which are appropriate in many situations, and investigate their expressive power. In a previous paper we have shown that the problem to determine the minimum schedule length is still intractable for this subclass, namely this problem is $\mathcal{NEXPTIME}$-complete as is the general case. Here we will investigate structural properties of the executions of such graphs. A natural graph-theoretic conjecture that executions must always split into components that are isomorphic to subgraphs turns out to be wrong. We are able to prove a weaker property. This implies a quadratic upper bound on the schedule length that may be necessary in the worst case, in contrast to the general case, where the optimal

* Corresponding author.
E-mail addresses: jakoby@tcs.uni-luebeck.de (A. Jakoby), liskiewi@tcs.uni-luebeck.de (M. Liśkiewicz), reischuk@tcs.uni-luebeck.de (R. Reischuk).

[1] Major part of this research was done while visiting the Department of Computer Science, University of Toronto.

[2] On leave from Instytut Informatyki, Uniwersytet Wrocławski, Poland.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2004.04.007

schedule length may be exponential with respect to the size of the representing DPG. Making this bound constructive, we obtain an approximation to a $\mathcal{NEXPTIME}$-complete problem. Computing such a schedule and then executing the program can be done on a parallel machine in polynomial time in a highly distributive fashion.

Keywords: Parallel processing; Dynamic process graphs; Scheduling; Overlap; Approximation

1. Introduction

Large parallel or distributed computations tend to have a lot of regularities. For example, the same instruction sequence may be executed by many processors in parallel. To describe the elementary steps and the logical dependencies among them one can use graphs, often called *process* or *data flow graphs*. One would like to keep the description of the parallel processes and their dependencies as compact as possible, for example not to unfold parallelism if this is not necessary. For this purpose, we have introduced a new graph model called *dynamic process graphs*, DPG for short. For a detailed motivation see [4].

These graphs possess an additional labelling function specifying the mode of the input and output behaviour of each node, where a node represents a process of the computation. This allows us to model basic primitives for specifying parallel programs, like *fork* and *join*. Following the OCCAM notion, these modes have been called ALT and PAR. If the *input mode* of a task v is ALT then in order to execute v *one* of the direct predecessor tasks has to be completed, whereas in case of PAR *all* of its direct predecessors have to be completed. A corresponding requirement is specified by the *output mode* when v has been finished. For the ALT output mode *one* of its direct successors (respectively *all* of them in case PAR) has to be initiated.

If one restricts the input and output mode to PAR then this variant is equivalent to ordinary data flow graphs. Using both modes, however, the representation of parallel programs by dynamic process graphs can provide an exponential compaction compared to the length of the actual execution sequences. Given a dynamic process graph, the first question that arises is whether it describes a legal parallel program. If yes then one would like to find an efficient execution of the program specified in such a compact form. We assume here that enough parallelism is available, so that the question turns into the problem of executing the program as fast as possible.

Dynamic process graphs and Boolean circuits are somehow related. We have shown that such graphs can be used to model computations of a circuit [4]. This has then been used to prove that dynamic process graphs using arbitrary combinations of modes provide a very compact nontrivial representation. Finding an optimal schedule, which is $\mathcal{NP}$-complete for ordinary graphs, turns out to be $\mathcal{NEXPTIME}$-complete. A similar complexity jump has been observed for classical graph problems in [1,6,7]. These papers have shown that simple graph properties become $\mathcal{NP}$-complete when the graph is represented in a particular succinct way using generating circuits or a hierarchical decomposition. Under the same representation graph properties that are $\mathcal{NP}$-complete, like HAMILTON CYCLE, 3-COLORABILITY, CLIQUE etc., become $\mathcal{NEXPTIME}$-complete.

If we put restrictions on the modes of a dynamic process graph its execution becomes easier. In [4,5] we have given a precise complexity classification for the different subclasses. The most interesting subclass seems to be DPGs for which the output mode is restricted to PAR, but for the input mode both alternatives are possible. Such graphs are still able to model, for example, the two natural ways in which objects of an object-oriented language can be activated: the total case, where all input parameters have to be specified before activation, and the partial case, where an object fires for any specified parameter once. For PAR-output graphs computing the minimum schedule length remains $\mathcal{NEXPTIME}$-complete [5]. This means that the search space of all possible solutions still has double exponential size.

This paper is concerned with approximations of optimal schedules for parallel programs specified by dynamic process graphs. Our main focus will be the maximal process duplication and the maximal overlap when executing different threads of programs. For the subclass of PAR-output graphs we prove a quadratic upper bound on their minimum schedule length. This contrasts to the unrestricted case for which this bound may be exponential. Thus, restricting the expressive power of this graph model leads to parallel and distributed programs with time complexity being decreased drastically. We will also show that an appropriate schedule can be constructed and executed time efficiently if enough processors are available. The optimal schedule can be approximated in a highly distributive fashion.

The rest of this paper is organized as follows. In the next section, we will give a formal definition of dynamic process graphs. Section 3 exhibits structural properties of this computational model. The upper bound on the approximation of an optimal schedule will be derived from a sequence of transformations described in Section 5. The distributive implementation is discussed in Section 7. We will conclude with some open problems.

2. Dynamic process graphs and runs

For a DAG, a directed acyclic graph $G = (V, E)$ with node set V and edge set E, let **pred**$(\boldsymbol{v})$ denote the set of direct (also called immediate) predecessors of a node $v \in V$ (the in-neighbourhood), and **succ**$(\boldsymbol{v})$ its direct successors (the out-neighbourhood). **pred***$(\boldsymbol{v})$ denotes the set of all predecessors of v, i.e., all nodes where v is reachable from. A *source* is a node with indegree 0, a *sink* has outdegree 0.

Definition 2.1. A *dynamic process graph* (DPG) $\mathcal{G} = (V, E, I, O)$ consists of a finite DAG (V, E) and two node labellings $I, O : V \to \{\mathsf{ALT}, \mathsf{PAR}\}$. $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of *processes* and E the dependencies among them. I and O describe *input*, (respectively *output*) *modes* of the v_i.

A *run* of $\mathcal{G}$ is a DAG $H_\mathcal{G} = (W, F)$ with a partition $W = W_1 \cup W_2 \cup \cdots \cup W_n$ of its node set. The nodes in W_i are called *execution instances* of process v_i. To express this correspondence W_i will also be denoted by $W(v_i)$. For some v_i the set $W(v_i)$ may be empty. However, a run $H_\mathcal{G}$ has to fulfill the following conditions:

(1) For each source node q of $\mathcal{G}$ the set $W(q)$ consists of exactly one execution instance.

(2) For every $v \in V$ with $\mathrm{pred}(v) = \{u_1, u_2, \ldots, u_p\} \neq \emptyset$, (respectively for each set $\mathrm{succ}(v) = \{u'_1, u'_2, \ldots, u'_r\} \neq \emptyset$), and for all $w \in W(v)$ it holds:
- if $I(v) = \mathsf{ALT}$ then w has a unique predecessor y belonging to $W(u_i)$ for some $i \in \{1, \ldots, p\}$;
- if $I(v) = \mathsf{PAR}$ then $\mathrm{pred}(w) = \{y_1, y_2, \ldots, y_p\}$ with $y_i \in W(u_i)$ for each $i \in \{1, \ldots, p\}$;
- if $O(v) = \mathsf{ALT}$ then w has a unique successor z belonging to $W(u'_j)$ for some $j \in \{1, \ldots, r\}$;
- if $O(v) = \mathsf{PAR}$ then $\mathrm{succ}(w) = \{z_1, z_2, \ldots, z_r\}$ with $z_j \in W(u'_j)$ for each $j \in \{1, \ldots, r\}$.

As size of a DPG $\mathcal{G}$ (respectively a run $H_{\mathcal{G}}$) we take the number of its nodes denoted by $|\mathcal{G}|$ (respectively $|H_{\mathcal{G}}|$).

A DPG $\mathcal{G}$ will be called *executable* if and only if there exist runs for it.

If not both types of modes occur in a DPG we get a restricted DPG. For example, a DPG with output mode $O \equiv \mathsf{PAR}$ will be called a PAR-*output* DPG, and if $I \equiv \mathsf{ALT}$ an ALT-*input* DPG.

As usual, our DAGs are not allowed to have multiple edges between a pair of nodes u, v. However, some constructions and illustrations given later in Section 5 will have several edges $e_1 = (u, v)$, $e_2 = (u, v)$, ... running in parallel. This is only done to simplify the presentation. To be formally correct, such edges can be distinguished by adding nodes in the middle, thus generating the edges $e'_1 = (u, \rho_1)$, $e''_1 = (\rho_1, v)$, $e'_2 = (u, \rho_2)$, $e''_2 = (\rho_2, v)$, ... Since these dummy nodes have indegree and outdegree 1 the input and output mode of these nodes are inessential. This local simplification will not have any influence on the essential properties of the DPGs considered there.

Fig. 1 shows a node v with input mode Q_1 and output mode Q_2. Throughout the paper we will illustrate the ALT-mode by a white box, the PAR-mode by a black box. In the following illustrations, for the convenience of the reader, the execution instances of a process will be grouped together in a shaded box.

Fig. 2 gives an example of a DPG and two runs of it. DPGs can be used to specify parallel programs in a compact way. A run then corresponds to an actual execution of the program, which can be smaller than its defining DPG. More typically, however, a run will be larger than the DPG itself since the PAR-constructor allows process duplications. The runs in Fig. 2 illustrate this property. We have shown an upper bound on the maximum size blow-up, respectively the possible compaction ratio of DPGs.

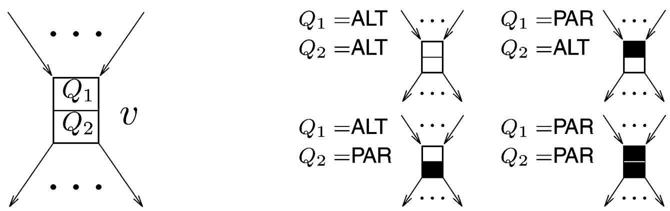

Fig. 1. A node v with input label Q_1 and output label Q_2 and the schematic representation.

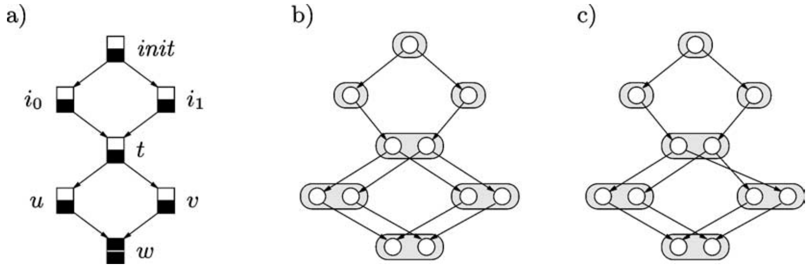

Fig. 2. (a) A DPG with 2 nonisomorphic runs: process t has 2 execution instances, and the same holds for all its successors; (b) a non-overlapping run, (c) a run with overlap at $W(t)$.

Theorem A [4]. *Let $\mathcal{G} = (V, E, I, O)$ be a DPG and $H_{\mathcal{G}} = (W, F)$ be a corresponding run. Then it holds $|W| \leqslant 2^{|V|-1}$, and this general upper bound is best possible.*

Thus, DPGs may have processes that require exponentially many execution instances with respect to the size of their description (an example illustrates Fig. 17).

The motivating question for our study of DPGs is *how efficiently a compactly specified parallel program Π can be executed.* Note that this is a crucial problem during compile-time. Furthermore, we like to construct an optimal schedule for the tasks of Π (compare, e.g., [2]). The result above implies an upper bound for the scheduling time of such a compactly represented program. However, this bound may be much too pessimistic. If enough processors are available it is conceivable that runs use only linear time or even less by executing many tasks in parallel. The maximal possible speedup obviously depends on the delay occurring when processors have to exchange data.

Definition 2.2. Let $\mathcal{G} = (V, E, I, O)$ be a DPG and $H = (W, F)$ be a run of $\mathcal{G}$. If $w \in W$ is an execution instance the *subrun* $\mathcal{R}(w)$ is the subgraph of H induced by $\mathbf{pred}^*(w)$, i.e., it consists of w and all its predecessors together with all their connections in F. We call a subrun $\mathcal{R}(w)$ of H *k-overlapping* iff for every node $v \in V$ it holds

$$\left|\mathcal{R}(w) \cap W(v)\right| \leqslant k + 1.$$

A run H is k-overlapping if all its subruns are k-overlapping. A 0-overlapping run will also be called *non-overlapping*. The overlap function for DPGs is given by

$$\gamma(\mathcal{G}) := \min\{k \mid \mathcal{G} \text{ has a } k\text{-overlapping run}\}.$$

We say that $\mathcal{G}$ requires k-overlap iff $\gamma(\mathcal{G}) \geqslant k$.

Note that non-overlapping implies that each subrun has to be isomorphic to a (not necessarily induced) subgraph of $\mathcal{G}$. This condition seems to be quite natural when executing tasks in parallel. Fig. 2, however, shows that runs may not be non-overlapping, even for DPGs with output restriction PAR. The two execution instances of the sink s induce two different subruns. For the run illustrated in (b) each subrun is non-overlapping, and it is isomorphic to a subgraph of $\mathcal{G}$. In (c) both subruns $\mathcal{R}(s)$ for the two execution instances corresponding to the sink s contain both execution instances of process t.

The maximum size of a subrun of H gives a better upper time bound for executing H than just the size of H because all subruns can be executed independently in parallel. Moreover, if a DPG $\mathcal{G}$ has a non-overlapping run then its execution instances can be executed in linear time with respect to the size of $\mathcal{G}$.

Now we give a formal definition for scheduling DPGs. It is based on scheduling its runs, which will be considered as ordinary precedence graphs. Let $\boldsymbol{\delta}$ be a parameter describing the communication delay of the system. To schedule a run $H = (W, F)$ with delay δ we assume that an unbounded number of processors can be used. In each unit-time interval a processor P can execute one single node w. In order to schedule w at time t each direct predecessor of w must have already been executed—either by P itself in previous time intervals or by some other processor by time interval $t - 1 - \delta$ such that the result of this predecessor can arrive at P on time. Scheduling precedence graphs in the presence of communication delays has first been considered in [8].

Definition 2.3. A schedule S for a DPG $\mathcal{G}$ with delay δ is a schedule of a run $H = (W, F)$ of $\mathcal{G}$. Let $T(S)$ denote the length of S, i.e., the point of time when S has executed all execution instances. The minimum schedule length of $\mathcal{G}$ is then given by

$$\boldsymbol{T}_{opt}(\mathcal{G}, \delta) := \min_{S \text{ schedule for } (\mathcal{G},\delta)} T(S).$$

To establish lower bounds, instead of the optimization problem to compute $T_{opt}(\mathcal{G}, \delta)$ and a schedule S achieving this bound we consider the decision problem.

Definition 2.4 (*Dynamic process graph schedule* (*DPGS*)). Given a DPG $\mathcal{G}$ with communication delay δ and a deadline T^*, does $T_{opt}(\mathcal{G}, \delta) \leqslant T^*$ hold?

The complexity of DPGS has been analysed in [4]. In general, the problem is difficult since runs may be of exponential size. The situation for computing an optimal schedule is even worse. The number of different runs can even grow double exponentially.

Theorem B [4]. *There exist families of DPGs with double exponential many different schedules with respect to the size of the underlying graphs.*

3. The structure of runs

In Theorem A we have seen that the size of a run can be exponential with respect to the size of its underlying DPG. But this does not imply that an optimal parallel execution of such a run requires exponential time. Each subrun may be small, thus exploiting enough parallelism the whole run may be completed quite fast.

The amount of overlap is a useful tool to estimate how efficiently a DPG can be executed in parallel. A simple thought shows that if $\mathcal{G} = (V, E, I, O)$ possesses a non-overlapping run then, independently of the communication delay, one can achieve $T_{opt}(\mathcal{G}, \delta) \leqslant |V|$. On the other hand, if $\mathcal{G}$ possesses a k-overlapping run then the following inequality holds: $T_{opt}(\mathcal{G}, \delta) \leqslant (k + 1) \cdot |V|$.

First let us investigate the question whether *non-overlapping* runs always exist for a given DPG.

3.1. Input restricted DPGs

Proposition 3.1. *For a DPG with a unique input mode, either $I \equiv \mathsf{ALT}$, or $I \equiv \mathsf{PAR}$, every run is non-overlapping.*

Proof. Let $\mathcal{G}$ be a DPG with run $H = (W, F)$. If only the ALT-input mode occurs for every $w \in W$ each subrun $\mathcal{R}(w)$ is a simple path connecting w with a source.

If $I \equiv \mathsf{PAR}$ then one can show by a simple topological induction starting from the sources that no process can have more than one execution instance: By definition, each source has exactly 1 execution instance. If $(u, v) \in E$ and $|W(v)| > 1$ then by definition of the PAR input mode it has to hold $|W(u)| > 1$, too. □

3.2. DPGs with output mode ALT

The non-overlapping property does not hold anymore when the input mode may vary—even if one restricts the output mode. In case of $O \equiv \mathsf{ALT}$, consider the simple graph in Fig. 3. It requires two execution instances for process v, and both have to be predecessors of the sink. However, we can show:

Proposition 3.2. *For ALT-output DPGs with κ sources it holds that every process has at most κ execution instances. Hence, the overlap can be at most $\kappa - 1$ and the size of a run is bounded by $\kappa \cdot |V|$.*

Proof. The upper bound on the number of execution instances can be shown by a flow-argument. Regard an ALT-output DPG $\mathcal{G}$ as a flow graph with unlimited edge capacity. We will construct a flow on $\mathcal{G}$ such that the flow leaving a node corresponds exactly to the number of execution instances this process requires. The requirement for the sources to have exactly one execution instance generates an initial flow of size κ from the sources to their direct successors. The ALT output mode implies that the flow does not increase. On the other hand, an internal node v with input mode PAR will decrease the flow by $|\mathrm{pred}(v)| - 1$. The flow leaving a node corresponds exactly to the number of execution instances this process requires. □

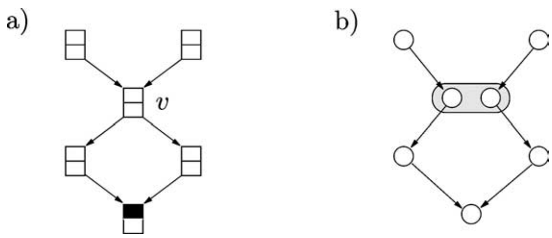

Fig. 3. A DPG with $\gamma(\mathcal{G}) > 0$: its unique run is 1-overlapping.

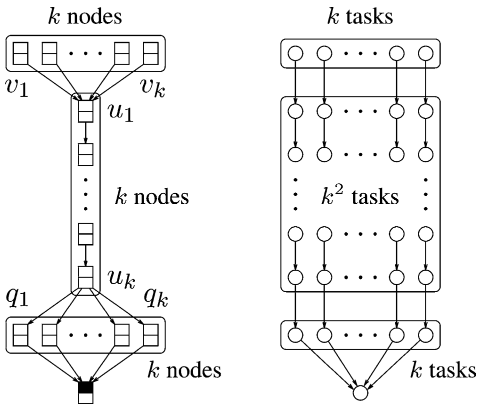

Fig. 4. Left: An ALT-output DPG that requires linear overlap; right: its unique run of quadratic size.

Since the number of sources is trivially bounded by the size of the DPG we get a linear upper bound on the overlap and a quadratic bound on the maximum size of a run.

To see that these estimates are asymptotically sharp consider the following family of DPGs $\mathcal{G}_1, \mathcal{G}_2, \ldots$ illustrated in Fig. 4. The nodes of $\mathcal{G}_k$ partition into three subsets $V_{k,1} = \{v_1, \ldots, v_k\}$, $V_{k,2} = \{u_1, \ldots, u_k\}$, $V_{k,3} = \{q_1, \ldots, q_k\}$, and a single sink s. The sink has PAR-input mode, all other nodes have ALT as input and output mode. The edges are given by:

$$E_k := \bigl\{(v_j, u_1), (u_k, q_j), (q_j, s) \mid j \in [1..k]\bigr\} \cup \bigl\{(u_j, u_{j+1}) \mid j \in [1..k-1]\bigr\}.$$

From the requirements of a run it is not difficult to verify that these DPGs have unique runs illustrated in Fig. 4 with linear overlap. Thus we can state the following result:

Proposition 3.3. *The family of* ALT*-output DPGs* $\mathcal{G}_k = (V_k, E_k, I_k, O_k)$ *of size* $3k + 1$ *defined above requires runs of size* $(k + 2) \cdot k + 1$ *and* $\gamma(\mathcal{G}_k) = k - 1$.

3.3. Unrestricted DPGs

By Proposition 3.2, in case of ALT output mode runs have a quadratic increase in size at most. In the unrestricted case, this property does not hold anymore as the following result shows.

Proposition 3.4. *For each natural number* k *there exists a DPG* $\mathcal{G}_k$ *of size* $2k + 1$ *with* $\gamma(\mathcal{G}_k) = 2^{k-1} - 1$. *Every run of* $\mathcal{G}_k$ *has size* $3 \cdot 2^{k-1}$.

Proof. $\mathcal{G}_k$ is constructed as follows. It consists of $2k + 1$ nodes $V_k := \{v_i \mid 1 \leqslant i \leqslant 2k + 1\}$ such that the first $k + 1$ nodes form a complete DAG as well as the last $k + 1$ nodes:

$$E_k := \bigl\{(v_i, v_j) \mid 1 \leqslant i < j \leqslant k + 1\bigr\} \cup \bigl\{ (v_i, v_j) \mid k + 1 \leqslant i < j \leqslant 2k + 1\bigr\}.$$

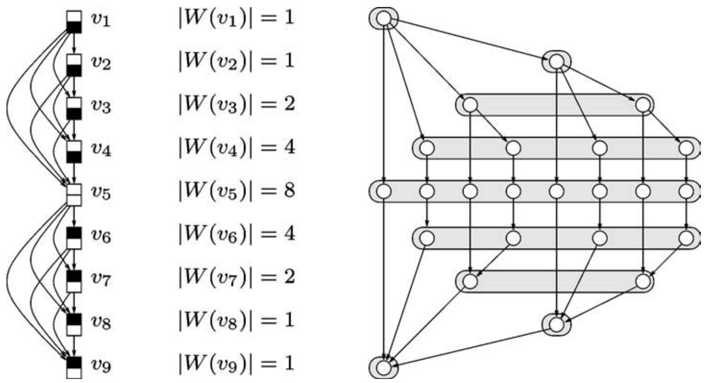

Fig. 5. Left: DPG $\mathcal{G}_4$; right: a corresponding run H_k. For $\delta_4 = 24$ the minimum schedule length is $T_{opt}(\mathcal{G}_4, \delta_4) = 24$.

The modes are given by

$$I(v_i) := \begin{cases} \mathsf{ALT} & \text{if } i \leqslant k+1, \\ \mathsf{PAR} & \text{if } i > k+1, \end{cases} \quad \text{and} \quad O(v_i) := \begin{cases} \mathsf{PAR} & \text{if } i < k+1, \\ \mathsf{ALT} & \text{if } i \geqslant k+1. \end{cases}$$

$\mathcal{G}_4$ and a run H_4 are shown in Fig. 5.

We claim that for every run H_k of $\mathcal{G}_k$ it holds

$$\left|W(v_i)\right| = \begin{cases} 1 & \text{for } i = 1, \\ 2^{i-2} & \text{for } 2 \leqslant i \leqslant k+1, \\ 2^{2k-i} & \text{for } k+1 \leqslant i \leqslant 2k, \\ 1 & \text{for } i = 2k+1. \end{cases} \tag{3.1}$$

Adding up these numbers we get that a run H_k requires $3 \cdot 2^{k-1} = 3 \cdot 2^{(|V|-1)/2-1}$ many execution instances, and the size of its middle layer $W(v_{k+1})$ equals 2^{k-1}. For the single execution instance w of the sink of $\mathcal{G}_k$, that is the unique node in $W(v_{2k+1})$ the subrun $\mathcal{R}(w)$ is identical to the complete run, thus all execution instances of the node v_{k+1} in the middle of $\mathcal{G}_k$ are predecessors of w. This proves that every run of $\mathcal{G}_k$ requires $(2^{k-1} - 1)$-overlap.

It remains to prove the correctness of Eq. (3.1). Let $\mathcal{G}_k^1$ be the subgraph of $\mathcal{G}_k$ consisting of the first $k+1$ nodes $\{v_1, \ldots, v_{k+1}\}$ and $\mathcal{G}_k^2$ be the subgraph of $\mathcal{G}_k$ consisting of the last $k+1$ nodes $\{v_{k+1}, \ldots, v_{2k+1}\}$. Note that the input and output mode of v_{k+1} is ALT. For any run of $\mathcal{G}_k^1$ it holds:

$$\left|W(v_i)\right| = \begin{cases} 1 & \text{for } i = 1, \\ 2^{i-2} & \text{for } 2 \leqslant i \leqslant k+1. \end{cases}$$

Assume that this equality is true for the graph $\mathcal{G}_{k-1}^1$. $\mathcal{G}_k^1$ consists of $\mathcal{G}_{k-1}^1$ followed by a single node v_{k+1}, which is a direct successor of any node v_i of $\mathcal{G}_{k-1}^1$. Because the input mode of v_{k+1} is ALT and the output mode of any node in $\mathcal{G}^1_{k-1}$ is PAR we can conclude

$$\left|W(v_{k+1})\right| = \sum_{i=1}^{k}\left|W(v_i)\right| = 1 + \sum_{i=2}^{k} 2^{i-2} = 2^{k-1}.$$

$\mathcal{G}^2_k$ consists of a single node v_{k+1} followed by a copy of $\mathcal{G}^2_{k-1}$, such that each node of $\mathcal{G}^2_{k-1}$ is a direct successor of v_{k+1}. We have to show that for any run of $\mathcal{G}^2_k$ it holds

$$\left|W(v_i)\right| = \begin{cases} 1 & \text{for } i = 2k+1, \\ 2^{2k-i} & \text{for } k+1 \leqslant i \leqslant 2k. \end{cases}$$

This is simply the property dual to $\mathcal{G}^1_{k-1}$, since every execution instance of $\mathcal{G}^2_{k-1}$ has a preceding execution instance in $W(v_{k+1})$. □

Hence, these DPGs require runs H_k of exponential size. In order to schedule such a run either an exponential number of processors or exponential time is necessary. In particular, since H_k possesses only one sink it does not pay to utilize more than 1 processor if the communication delay δ_k is large—here $\delta_k = |H_k| = 3 \cdot 2^{k-1}$ suffices.

Corollary 3.5. *For the family $\mathcal{G}_k$ defined above it holds $T_{opt}(\mathcal{G}_k, \delta_k) = 3 \cdot 2^{(|V|-1)/2-1}$ for large communication delay.*

Let us remind that communication delay complicates the scheduling problem with an unlimited number of processors significantly only if the delay grows with the size of the graphs [3].

4. PAR-output DPGs

In the rest of this paper we will analyse the remaining case of DPGs with output mode restricted to PAR. Contrary to the previous cases, there does not seem to be an easy argument how the size of their runs and their scheduling time can be bounded. We will achieve this goal by estimating the maximum overlap necessary. As one can see in Fig. 2(c) PAR-output DPGs may have runs that are overlapping. Fig. 2(b) shows that this overlapping run can be modified into a non-overlapping one. Such a property, however, does not hold in general; in other words, there exist DPGs $\mathcal{G}$ with $\gamma(\mathcal{G}) > 0$.

As the main technical result of this paper we will establish a nontrivial upper bound on $\gamma(\mathcal{G})$. The proof will be constructive. From it we can deduce an efficient approximation method for computing an optimal schedule. Furthermore, this approach allows a fast parallel implementation.

In the following, only for the purpose of simplifying the presentation, some DPGs will be drawn with *multiple edges* between two nodes u, v. Instead of using multiple edges we could differentiate such edges by splitting them and adding an additional node in the middle. This, however, would only blow-up the construction.

In general it will not be necessary to make an explicit distinction between multiple edges between two nodes u, v. If necessary we will use subscripts to uniquely label them: $(u, v)_0, (u, v)_1, \ldots.$

Let us first discuss some general properties of PAR-output DPGs. For a run $H_\mathcal{G} = (W, F)$ of $\mathcal{G}$ with node partition $W = \bigcup_{i=1}^{n} W(v_i)$ define the *characteristic vector* as the sequence

$$\chi(H_\mathcal{G}) := \left(\left|W(v_1)\right|, \left|W(v_2)\right|, \ldots, \left|W(v_n)\right|\right),$$

and $\mu(H_\mathcal{G})$ as the maximal value of the entries in $\chi(H_\mathcal{G})$, that means $\mu(H_\mathcal{G}) := \max_{v \in \mathcal{G}} |W(v)|$. By Theorem A, for all i holds:

$$\left|W(v_i)\right| \leqslant |W| \leqslant 2^{|V|-1},$$

thus $\mu(H_\mathcal{G}) \leqslant 2^{|V|-1}$.

Lemma 4.1. *For a* PAR*-output DPG $\mathcal{G}$ all runs have exactly the same characteristic vector.*

Proof. For any PAR-output executable DPG $\mathcal{G}$ the following breadth-first search procedure computes the characteristic vector independently of a particular run of $\mathcal{G}$. For every source node v, by definition it has to hold $|W(v)| = 1$. Then for a non-source node v with $I(v) =$ ALT, by the conditions of a run one has to choose

$$\left|W(v)\right| = \sum_{v_j \in \mathrm{pred}(v)} \left|W(v_j)\right|$$

many execution instances. For v with $I(v) =$ PAR all direct predecessors $u_1, \ldots, u_k$ of v must have the same number of execution instances and $|W(v)|$ has to equal this number—otherwise $\mathcal{G}$ would not be executable. □

This agreement between different runs of $\mathcal{G}$ implies that for the characteristic vector instead of $\chi(H_\mathcal{G})$ we can simply write $\chi(\mathcal{G})$, and similarly $\mu(\mathcal{G})$.

Corollary 4.2. $\mu(\mathcal{G}) \leqslant 2^{|\mathcal{G}|-1}$.

Theorem 4.3. *There exists an executable* PAR*-output DPG $\mathcal{G}$ with $\gamma(\mathcal{G}) > 0$.*

Proof. We construct a PAR-output DPG $\mathcal{G} = (V, E, I, O)$ as follows. Let $V := \{a, b, c, d, e, f, g, h, i, j, k\}$ and E be defined as in Fig. 6. Note that $\mathcal{G}$ has multiple edges (e.g., between a and b). Let $I(v) :=$ PAR for $v \in \{a, f, k\}$, and the remaining nodes have ALT input mode.

We claim that for this graph $\gamma(\mathcal{G}) > 0$, i.e., every run of $\mathcal{G}$ requires nontrivial overlapping. To see this let us assume to the contrary that $H_\mathcal{G} = (W, F)$ is a non-overlapping run of $\mathcal{G}$. The unique characteristic vector of $\mathcal{G}$ is $(1, 2, 3, 4, 4, 4, 6, 8, 12, 12, 12)$. Let $W(a) = \{u_0\}$, $W(b) = \{v_1, v_2\}$, $W(c) = \{w_1, w_2, w_3\}$, and name the execution instances such that

$$(u_0, v_1),\ (u_0, v_2),\ (v_1, w_1),\ (v_2, w_2),\ (u_0, w_3) \in F.$$

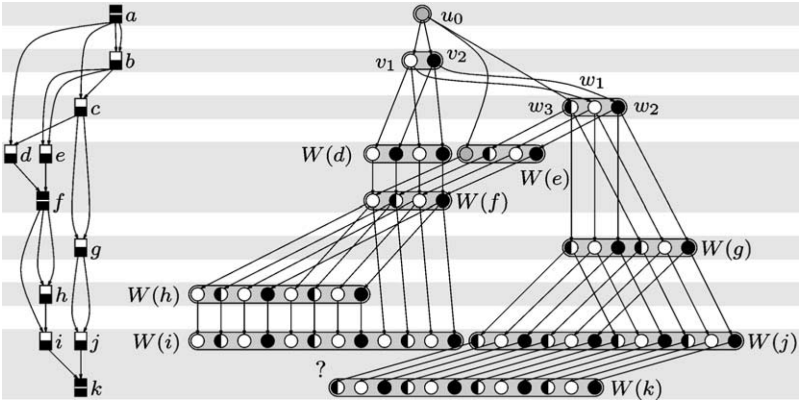

Fig. 6. Left: Graph $\mathcal{G}$ with $\gamma(\mathcal{G}) > 0$; right: a fragment of a run $H_{\mathcal{G}}$. To facilitate the understanding, nodes with the same number of execution instances are grouped together in horizontal stripes.

We colour the nodes of $H_{\mathcal{G}}$ using four colours 0, 1, 2, 3 as follows:

$$C(u_0) := 0, \quad C(v_1) := C(w_1) := 1, \quad C(v_2) := C(w_2) := 2, \quad C(w_3) := 3.$$

The colouring of nodes in $W(g)$ and in $W(h)$ will be inessential, therefore we omit to specify them. For each node $x \in W(j)$ define $C(x) := \ell$ if x is a successor of w_ℓ. Hence, a multiset describing the colours of the execution instances in $W(j)$ is equal to

$$\{1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3\}. \tag{4.1}$$

For any $x \in W(d) \cup W(e)$ let $C(x)$ be equal to the colour of its direct predecessor. Moreover, for any $x \in W(f)$, if $z \in W(d)$ and $y \in W(e)$ denote its direct predecessors then one of the following four cases has to occur (otherwise $H_{\mathcal{G}}$ would be 1-overlapping):

(1) $C(y) = 1$ and $C(z) = 1$,
(2) $C(y) = 2$ and $C(z) = 2$,
(3) $C(y) = 3$ and $C(z) \in \{1, 2\}$,
(4) $C(y) = 0$ and $C(z) \in \{1, 2\}$.

In case (1), (2) and (3) let $C(x) := C(y)$; in case (4) let $C(x) := C(z)$. Hence, the multiset describing the colours of $W(f)$ is equal either to $\{1, 1, 2, 3\}$ or to $\{1, 2, 2, 3\}$. For $x \in W(i)$ let $C(x)$ be equal to the colour of its predecessor in $W(f)$. This implies that the multiset of colours of $W(i)$ is either

$$\{1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3\} \quad \text{or} \quad \{1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3\}. \tag{4.2}$$

Obviously, we get that for any $x \in W(j)$ node $w_{C(x)}$ is a predecessor of x, and additionally if $C(x) \in \{1, 2\}$ then also $v_{C(x)}$ is a predecessor of x. Moreover, for any $y \in W(i)$ it holds that if $C(y) \in \{1, 2\}$ then node $v_{C(x)}$ has to be a predecessor of y; if $C(y) = 3$ then it is required that w_3 is its predecessor. Now consider the connections between $W(k)$ and $W(i)$, $W(j)$ in $H_{\mathcal{G}}$. From the remark above it follows that if $x \in W(j)$ and $y \in W(i)$ are direct predecessors of $z \in W(k)$ then it holds

$$C(x) = 1 \to C(y) = 1 \quad \text{and} \quad C(x) = 2 \to C(y) = 2.$$

These conditions and the multiplicities of colours (4.1) imply that for $\ell = 1, 2$ it holds

$$\left|\{y \in W(i)\colon C(y) = \ell\}\right| \geqslant \left|\{x \in W(j)\colon C(x) = \ell\}\right| = 4.$$

But according to (4.2) this is impossible, thus we get a contradiction. □

The DPG in Fig. 6 and its runs already look quite complicated. It was not easy to find such an example of a graph with nonzero overlap, and it seems that this construction cannot be iterated to establish a better lower bound. Thus, the best lower bound on the overlap of PAR-output DPGs known to us at the moment is only 1. On the other hand, establishing good upper bounds also turned out to be quite difficult. By a lengthy and complicated construction we have been able to prove the following nontrivial bound.

Theorem 4.4. *For every executable PAR-output DPG $\mathcal{G}$ it holds $\gamma(\mathcal{G}) \leqslant \log_2 \mu(\mathcal{G})$.*

The proof of this theorem will be given in Section 6. For each fixed μ-value we will construct a *most complicated* DPG *with respect to overlapping* and establish an upper bound on its overlap.

From Theorem 4.4 and the exponential upper bound on the maximum size of a run $\mu(\mathcal{G}) \leqslant 2^{|\mathcal{G}|-1}$ we can conclude $\gamma(\mathcal{G}) \leqslant |\mathcal{G}| - 1$, i.e.:

Corollary 4.5. *Every executable PAR-output DPG has a run with an overlap that is linearly bounded with respect to its size.*

As indicated above, this result implies a nontrivial upper bound for scheduling PAR-output DPGs.

Corollary 4.6. *For every executable PAR-output DPG $\mathcal{G}$ it holds $T_{opt}(\mathcal{G}, \delta) \leqslant |\mathcal{G}|^2$, independently of the communication delay δ.*

Hence, PAR-output DPGs can be executed quite fast. Our proof will be constructive and yields an efficient method to generate a schedule S for $\mathcal{G}$ with $T(S) \leqslant |\mathcal{G}|^2$.

We obtain this upper bound for the minimum schedule length by considering $\gamma(\mathcal{G})$-overlapping runs for $\mathcal{G}$. It may be conjectured that runs with minimum overlap can always be scheduled in optimal time $T_{opt}(\mathcal{G}, \delta)$. But we can show that this is not the case.

Theorem 4.7. *There exist* PAR*-output DPGs $\mathcal{G}$ with $\gamma(\mathcal{G}) = 0$ such that for an appropriately chosen communication delay δ no non-overlapping run of $\mathcal{G}$ can achieve the minimum schedule length $T_{opt}(\mathcal{G}, \delta)$. In other words, overlap between execution instances may be necessary for optimal schedules even if the DPG can be executed without overlap.*

Proof. Fig. 7 illustrates such a family of DPGs. Run (a) in the middle is non-overlapping, run (b) on the right is 1-overlapping. The sequence A is the chain of execution instances for

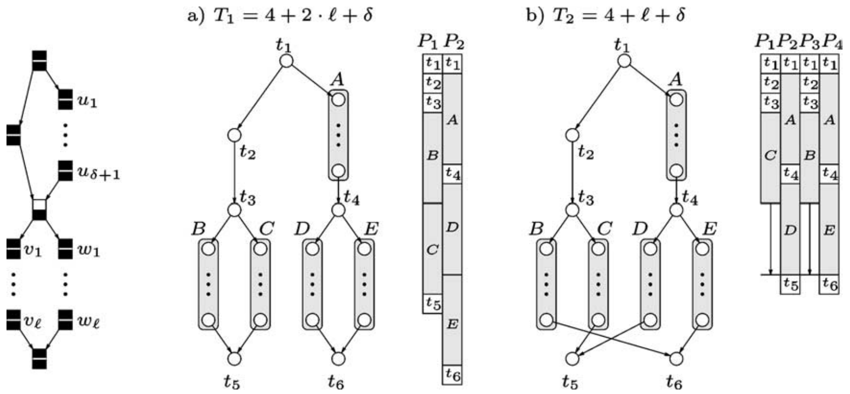

Fig. 7. Left: A DPG $\mathcal{G}$ parameterized by 2 natural numbers ℓ and δ. Middle and right: Two different runs (a) and (b) of $\mathcal{G}$; the appropriate optimal schedules are illustrated on the right.

$u_1, \ldots, u_{\delta+1}$, similarly B and D for $v_1, \ldots, v_\ell$, and C and E for $w_1, \ldots, w_\ell$. If $\delta \geqslant \ell$ and the communication delay is at least as large as δ, a minimum schedule in case (a) requires time $T_1 = 4 + 2\ell + \delta$, in case (b) time $T_2 = 4 + \ell + \delta$. $\square$

5. Transforming PAR-output DPGs into a normal form

Our overall proof strategy to approximate the minimum schedule length based on the notion of overlap can be described as follows. We define a special family of PAR-output DPGs that have a regular structure called *normal form.* It will be shown that these DPGs require the largest overlap. This is done by proving that an arbitrary PAR-output DPG can be *embedded* into a normal form DPG without increasing the number of executions of its sinks. Furthermore, a run for a normal form DPG can be transformed into a run of the original DPG without increasing the overlap. Finally, for DPGs in this normal form we are able to establish an upper bound on the overlap. Their regular structure allows us to make an inductive construction that generates runs with small overlap.

This section describes the transformation of PAR-output DPGs into a normal form DPG. It will be quite technical. For DPGs $\mathcal{G}$ of this special form we will establish the upper bound on their maximum overlap $\gamma(\mathcal{G})$ in the following section. Combining this bound with a reverse transformation for corresponding runs from the normal form to an arbitrary graph and ensuring that the overlap does not increase during this procedure the upper bound on $\gamma(\mathcal{G})$ for arbitrary $\mathcal{G}$ as stated in Theorem 4.4 will be obtained.

5.1. DPGs with synchronization

Given an executable PAR-output DPG $\mathcal{G}$, to construct a $\gamma(\mathcal{G})$-overlapping run seems difficult. To warm up, let us first consider a restricted class of PAR-output DPGs.

Definition 5.1. A PAR-output DPG is called *synchronized* if each nonsource node has indegree 2, all paths from a source to a sink have the same length, and for each such path the sequence of input modes of its nodes alternates between PAR and ALT.

First, it will be proven that such graphs have non-overlapping runs. This will be our starting point to generalize the proof techniques to arbitrary PAR-output DPGs.

Proposition 5.2. *Every synchronized* PAR-*output DPG $\mathcal{G}$ has a non-overlapping run. Hence, it holds $\gamma(\mathcal{G}) = 0$ and $T_{opt}(\mathcal{G}, \delta) \leqslant |V|$, independently of the communication delay δ.*

Proof. By induction, assume that this claim holds for every synchronized DPG of depth ℓ (the base case $\ell = 1$ being trivial), and let $\mathcal{G}$ be a synchronized DPG of depth $\ell + 1$. Let $\mathcal{G}'$ be the DPG that we get from $\mathcal{G}$ by cutting off all sources of $\mathcal{G}$. Note that $\mathcal{G}'$ is a synchronized DPG of depth ℓ. By the inductive assumption, there exists a non-overlapping run $H_{\mathcal{G}'}$ for $\mathcal{G}'$. Furthermore, let $L_0 := \{q_1, \dots, q_k\}$ denote the set of sources of $\mathcal{G}$, and L_1 the set of nodes which are connected directly to these sources. Then we can generate a non-overlapping run $H_{\mathcal{G}} = (W, F)$ for $\mathcal{G}$ as follows.

(1) If the input mode of the nodes in L_1 is PAR then by construction of $\mathcal{G}'$, in $H_{\mathcal{G}'}$ there exists exactly one execution instance $t_i \in W(v_i)$ for every node $v_i \in L_1$. The run $H_{\mathcal{G}}$ consists of a copy of $H_{\mathcal{G}'}$ and an additional execution instance w_i for each $q_i \in L_0$. Then $(w_j, t_i) \in F$ iff $(q_j, v_i) \in E$. It is obvious that $H_{\mathcal{G}}$ is a run for $\mathcal{G}$. Since $H_{\mathcal{G}'}$ is non-overlapping and there exists only one execution instance for each node in L_0, the resulting run $H_{\mathcal{G}}$ is also non-overlapping.
(2) If the input mode of the nodes in L_1 is ALT let E_1 and E_2 be a partition of the edges in $E \cap (L_0 \times L_1)$ such that for every $v_i \in L_1$ one of its incoming edges belongs to E_1 and the other to E_2 (recall that by the synchronization property each $v_i \in L_1$ has indegree 2). Now $H_{\mathcal{G}}$ consists of two copies of $H_{\mathcal{G}'}$, say $H^1_{\mathcal{G}}$ and $H^2_{\mathcal{G}}$, and additional execution instances w_i for nodes $v_i \in L_0$. If $t^1_i \in H^1_{\mathcal{G}}$ is an execution instance of a source node v_i of $\mathcal{G}'$ then we draw an edge from w_j to t^1_i iff $(u_j, v_i) \in E_1$. Analogously, we draw an edge between w_j and a sink t^2_i of $H^2_{\mathcal{G}}$ iff $(u_j, v_i) \in E_2$. This yields a valid run $H_{\mathcal{G}}$ for $\mathcal{G}$. Furthermore, if $H_{\mathcal{G}'}$ is non-overlapping and there exists only one execution instance for each node $u \in L_0$, the resulting run $H_{\mathcal{G}}$ is non-overlapping, too. $\square$

Now consider an arbitrary executable PAR-output DPG $\mathcal{G} = (V, E, I, O)$. The transformation of $\mathcal{G}$ consists of five stages:

$$\mathcal{G} \xrightarrow{(A)} \boldsymbol{\mathcal{G}_A} \xrightarrow{(B)} \boldsymbol{\mathcal{G}_B} \xrightarrow{(C)} \boldsymbol{\mathcal{G}_C} \xrightarrow{(D)} \boldsymbol{\mathcal{G}_D} \xrightarrow{(E)} \boldsymbol{\mathcal{G}_E}.$$

Each stage puts an additional restriction on the graphs as follows:

(A) every node with input mode ALT has indegree exactly two;
(B) every direct predecessor and every direct successor of a node with input mode PAR has input mode ALT;

(C) the sources and sinks have input mode PAR and on every path from a source to a sink the input mode of the nodes alternates;
(D) for each value $i \leqslant \mu(\mathcal{G})$ there exists at most one PAR-input node that requires exactly i execution instances. This unique node will be called $\boldsymbol{u_i}$ in the following;
(E) for each pair of numbers $1 \leqslant i \leqslant j$ with $i + j \leqslant \mu(\mathcal{G})$ there exists at most one ALT-input node $\boldsymbol{v_{i,j}}$ requiring exactly $i + j$ execution instances such that $(u_i, v_{i,j}), (u_j, v_{i,j})$ are edges of $\mathcal{G}_E$, where u_i, u_j are the unique nodes with i (respectively j) execution instances. This includes the special case $i = j$ in which there are two edges that run from u_i to $v_{i,j}$.

These transformations will not change the μ-value of the DPGs—the maximal number of execution instances a process requires. Furthermore, we will make sure that if $\mathcal{G}_E$ has a $\gamma(\mathcal{G}_E)$-overlapping run then the graph of each stage has a $\gamma(\mathcal{G}_E)$-overlapping run, too. More precisely,

$$\gamma(\mathcal{G}) \leqslant \gamma(\mathcal{G}_A) \leqslant \gamma(\mathcal{G}_B) \leqslant \gamma(\mathcal{G}_C) \leqslant \gamma(\mathcal{G}_D) \leqslant \gamma(\mathcal{G}_E).$$

5.2. Stage A: indegree exactly 2

As the first step the indegree of each node v of $\mathcal{G}$ with $I(v) = \mathsf{ALT}$ is reduced by replacing it by an appropriate binary tree in $\mathcal{G}_A$. From a $\gamma(\mathcal{G}_A)$-overlapping run H_A of $\mathcal{G}_A$ we show how to construct a $\gamma(\mathcal{G}_A)$-overlapping run H of $\mathcal{G}$. To obtain H from H_A one only needs to delete all execution instances of nodes of the auxiliary binary trees except for the roots and then connect the roots with the remaining execution nodes in the obvious way.

The details of this transformation $\mathcal{G} \xrightarrow{\text{(A)}} \mathcal{G}_A$ are as follows. Since every node with only one direct predecessor can be assigned input mode PAR, we can assume that each node with input mode ALT has indegree at least 2. In the new DPG $\mathcal{G}_A = (V_A, E_A, I_A, O_A)$ each ALT-input node will have indegree exactly 2. Every $v \in V$ has a counterpart $g_A(v)$ in V_A—which for short will be denoted by v'—with the same input and output mode. For every edge (u, v) such that $I(v) = \mathsf{ALT}$ with *indegree*$(v) = 2$, or $I(v) = \mathsf{PAR}$, we add the edge $(u', v') = (g_A(u), g_A(v))$ to E_A. For each ALT-input node $v \in V$ with indegree $d > 2$ let $v_1, v_2, \ldots, v_d$ be the sequence of its direct predecessors i.e., $(v_1, v) \ldots (v_k, v)$ are edges of E. Then we add $2d - 2$ nodes $u_v^1, u_v^2, \ldots, u_v^{2d-2}$ to V_A—in the following called *dummy nodes*—and connecting edges to E_A such that we obtain a binary tree rooted in v', with $u_v^1, u_v^2, \ldots, u_v^{d-2}$ as inner nodes, and $u_v^{d-1}, u_v^d, \ldots, u_v^{2d-2}$ as leaves. As input mode for the inner nodes we chose ALT, while for the leaves the mode is set to PAR. Next, we connect these leaves with the nodes $g_A(v_i)$ by drawing for each $i \in [1..d]$ an edge from $g_A(v_i)$ to u_v^{d+i-2}. The transformation of $\mathcal{G}$ is illustrated in Fig. 8(a). We extend the mapping g_A to the edges of $\mathcal{G}$ by

$$g_A : V \cup E \to V_A \cup E_A$$

as follows: for any edge (u, v) if $I(v) = \mathsf{PAR}$ or $I(v) = \mathsf{ALT}$ and *indegree*$(v) = 2$ then let $g_A(u, v) := (u', v')$. Otherwise, for any direct predecessor v_i of v let $g_A(v_i, v) = (g_A(v_i), u)$, where u is the direct successor of $g_A(v_i)$ in the tree with root v'.

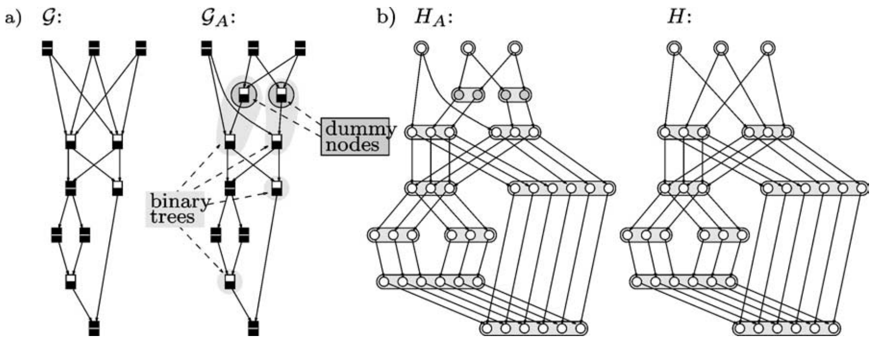

Fig. 8. (a) A PAR-output DPG before and after transformation A; (b) the corresponding runs.

In general, for a mapping $g : V \cup E \to V' \cup E'$ between graphs $G = (V, E)$ and $G' = (V', E')$ we define an inverse partial function

$$\bar{g} : V' \cup E' \to 2^{V \cup E}$$

by $\bar{g}(x) := \{z \in V \cup E \mid g(z) = x\}$. In the special case of the function g_A since it is injective its inverse $\bar{g}_A$ can be considered as a partial function with range $V \cup E$.

We generate a run $H := (W, F)$ for $\mathcal{G}$ by simply deleting the execution instances of dummy nodes from the run $H_A := (W_A, F_A)$ of $\mathcal{G}_A$ and connect the execution instances of the nodes $g_A(v)$ directly as described by the dummy execution instances (see Fig. 8(b)). Since the dummy nodes only introduce simple paths in the run of $\mathcal{G}_A$, the resulting graph is a run for $\mathcal{G}$. Note that this modification can be performed for each subrun independently. Let us call a path *dummy* if all its nodes except the endpoints are dummy nodes. Then a formal definition can be given by

$$W := \bigcup_{v \in V} W_A\big(g_A(v)\big),$$

$$F := \big\{(s, t) \in W \times W \mid (s, t) \in F_A \text{ or there exists a dummy path from } s \text{ to } t \text{ in } H_A\big\}.$$

This operation does not change the maximal number of execution instances required to execute the DPG. Furthermore, if the run of $\mathcal{G}_A$ is k-overlapping the same holds for the run of $\mathcal{G}$ obtained this way. Hence, we have shown:

Lemma 5.3. $\mu(\mathcal{G}) = \mu(\mathcal{G}_A)$ *and* $\gamma(\mathcal{G}) \leqslant \gamma(\mathcal{G}_A)$.

For later implementation of this transformation let us add the following remark. Given a run H_A of $\mathcal{G}_A$ we can compute a run H for $\mathcal{G}$ by using $\bar{g}_A$ and deleting all the dummy nodes and edges starting at a dummy node locally. Hence, this operation can be performed at each node and each edge of H_A *independently in parallel.*

5.3. *Stage B: with respect to the input mode,* PAR*-nodes are surrounded by* ALT*-nodes*

If we remove from $\mathcal{G}_A$ all nodes with input mode ALT it will fall apart into a bunch of connected components $C_1, C_2, \ldots$ that contain only PAR-input nodes. With respect to the execution problem a component C_i almost behaves like an ordinary process graph since both input and output are fixed to PAR. In particular, all its nodes require the same number of execution instances. Thus, we can shrink each such component C_i to a single node c_i to obtain a reduced DPG $\mathcal{G}_B = (V_B, E_B, I_B, O_B)$ (see Fig. 9(a)).

Speaking more formally, any $v \in V_A$ with $I_A(V) = \mathsf{ALT}$ possesses a direct counterpart $v' = g_B(v)$ in V_B. In addition, V_B contains a node c_i for each component C_i. For nodes with PAR input mode let $K(v)$ denote the index i of the component C_i that contains v and define $g_B(v) = c_{K(v)}$. E_B is defined as follows. Assume that E_A contains an edge $e = (u, v)$ and let $u' = g_B(u)$ and $v' = g_B(v)$. Then one of the following four cases occurs:

(1) $I_A(u) = I_A(v) = \mathsf{ALT}$: then E_B contains the edge $e' := (u', v')$;
(2) $I_A(u) = \mathsf{ALT}$ and $I_A(v) = \mathsf{PAR}$: then add the edge $e' := (u', c_{K(v)})$ to E_B. If there are other nodes $\tilde{v}$ in the component $C_{K(v)}$ all these edges $(u, \tilde{v})$ are represented by the same edge e' in $\mathcal{G}_B$.
(3) $I_A(u) = \mathsf{PAR}$ and $I_A(v) = \mathsf{ALT}$: then add the edge $e' := (c_{K(u)}, v')$ to E_B. Since v has indegree 2 there exists another edge $(\tilde{u}, v)$ with endpoint v. However, if $\tilde{u}$ is also a PAR-input node and $K(u) = K(\tilde{u})$ then we add two edges $e'_0 := (c_{K(u)}, v')_0$ and $e'_1 := (c_{K(\tilde{u})}, v')_1$ to E_B. These two edges are numbered according to an ordering between u and $\tilde{u}$ (we can assume that a total order on the nodes of $\mathcal{G}_A$ is given—for example, deduced from their names).
(4) $I_A(u) = I_A(v) = \mathsf{PAR}$: this case does not add any edge to E_B.

In the first three cases we define $g_B(e) := e'$, in the last one $g_B(e) := c_{K(v)}$.

From a k-overlapping run $H_B = (W_B, F_B)$ for $\mathcal{G}_B$ a k-overlapping run for $\mathcal{G}_A$ can be generated by replacing each execution instance z_i of a node c_i by a copy Z_i of its corresponding component C_i in $\mathcal{G}_A$. The nodes of such a copy Z_i are connected to the predecessors of z_i in H_B according to the topology of $\mathcal{G}_A$. To connect these nodes to their successors is slightly more involved. Let Γ_i denote the set of direct successors of z_i. For every edge $(u, v) \in E_A$ there is a unique edge $(c_{K(u)}, v) \in E_B$. Furthermore, for every pair of execution instances $z_{K(u)} \in W(c_{K(u)})$ and $t \in W(v)$ and every edge $(c_{K(u)}, v)$ there exists an edge $(z_{K(u)}, t)$ in H_B. Hence, we can use this relation to connect the nodes of Z_i to the execution instances of v according to the topology of $\mathcal{G}_A$. The bound on the overlap in H_B implies the same bound for the resulting run of $\mathcal{G}_A$ and the maximal number of execution instances does not change. The relations between the runs of $\mathcal{G}_A$ and $\mathcal{G}_B$ are illustrated in Fig. 9(b).

Thus, it holds

Lemma 5.4. *$\mu(\mathcal{G}_A) = \mu(\mathcal{G}_B)$ and $\gamma(\mathcal{G}_A) \leqslant \gamma(\mathcal{G}_B)$.*

The inverse mapping $\bar{g}_B$ is also a local operation on the graph $\mathcal{G}_B$. Moreover given a run H_B for $\mathcal{G}_B$ we can compute a run H_A for $\mathcal{G}_A$ by performing a mapping based on $\bar{g}_B$ on each element of H_B independently in parallel.

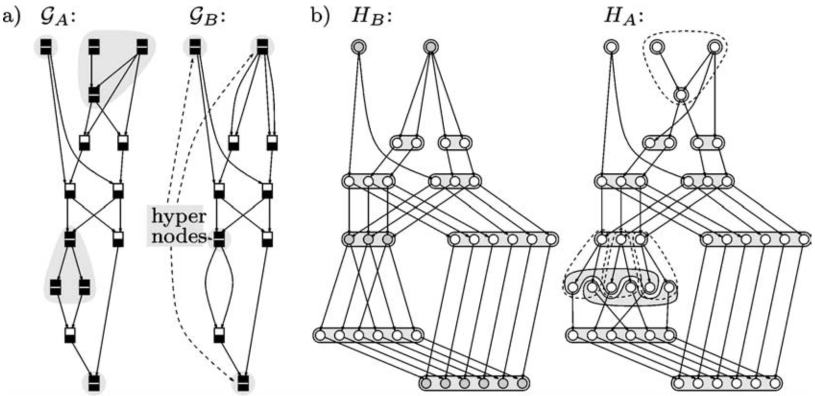

Fig. 9. (a) A PAR-output DPG before and after transformation B; (b) the corresponding runs.

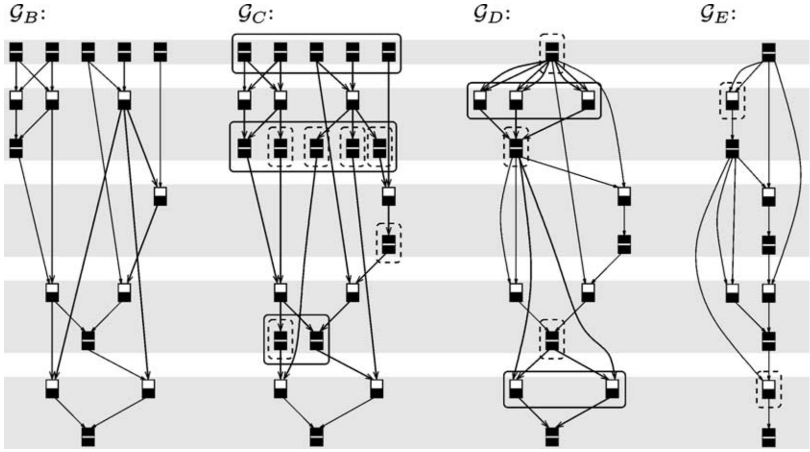

Fig. 10. A PAR-output DPG after four successive transformations: B, C, D, and finally E. Oval frames in the illustration above indicate nodes which will be transformed in the next stage. The dashed oval frames indicate the resulting new nodes of a particular transformation.

5.4. *Stage C: the input mode alternates; sources and sinks have input mode* PAR

To achieve property (C) it suffices to add dummy nodes with PAR-input mode to $\mathcal{G}_B$ between successive nodes that have the same input mode, which, due to the previous transformation, can only be the ALT-mode. In addition, we can make sure that the input mode of each source and each sink is PAR. This transformation will be denoted by $g_C(\mathcal{G}_B) := \mathcal{G}_C$—an example is illustrated in Fig. 10. Note that this transformation increases the size of the graph by a factor of at most two. To generate a run of $\mathcal{G}_B$ using a run H_C of $\mathcal{G}_C$ we delete the execution instances of the additional PAR-input nodes and connect their predecessors and successors directly—(see Fig. 11 for an example). It follows directly:

Lemma 5.5. *$\mu(\mathcal{G}_B) = \mu(\mathcal{G}_C)$ and $\gamma(\mathcal{G}_B) \leqslant \gamma(\mathcal{G}_C)$.*

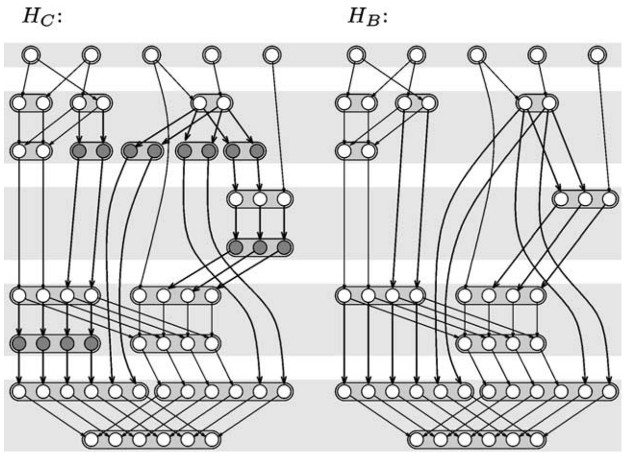

Fig. 11. Transformation from a run H_C to run H_B.

As in stage A, the inverse function $\bar{g}_C$ can be computed by a simple local operation on $\mathcal{G}_C$. Given a run H_C for $\mathcal{G}_C$ we can compute a run H_B for $\mathcal{G}_B$ by using $\bar{g}_C$ and deleting all the dummy nodes and edges. As in the previous stages this operation can be performed on each execution node and edge of H_C independently in parallel.

After having performed this transformation one can divide the nodes of $\mathcal{G}$ into levels as indicated by the gray horizontal sections in Fig. 10. In every level all its nodes have the same number of execution instances. Since the input mode alternates and ALT-input nodes increase the number of execution instances, whereas for PAR-input nodes this number stays constant, each level has depth at most 2. We define these levels as follows: $\mathcal{L}_1(\mathcal{G})$ contains the sources, which all have input mode PAR, and $\mathcal{L}_i(\mathcal{G})$ for $i > 1$ the level such that the number of execution instances of a process is i. Let $\mathcal{L}_i^1(\mathcal{G})$ denote the nodes of level $\mathcal{L}_i$ with input mode ALT, and $\mathcal{L}_i^2(\mathcal{G})$ those with input mode PAR. For $i > 1$, certain $\mathcal{L}_i(\mathcal{G})$ may be empty. But due to the alternation property, if $\mathcal{L}_i(\mathcal{G})$ is nonempty then both its sublevels $\mathcal{L}_i^j(\mathcal{G})$ with $j = 1, 2$ are nonempty.

5.5. *Stage D: eliminating* PAR-*input duplicates*

The next step reduces the number of PAR-input nodes at each nonempty level $\mathcal{L}_i$ to a single node such that for any value $i \leqslant \mu(\mathcal{G})$ there exists at most one PAR-input node with i execution instances called u_i. To construct the new DPG $\mathcal{G}_D$, for every node $v \in V_C$ with $I(v) = \mathsf{ALT}$ we generate a node $v' = g_D(v)$. For every nonempty PAR-input level $\mathcal{L}_i^2$ of $\mathcal{G}$, a PAR-input node c_i is generated, and $g_D(v) := c_i$ for all $v \in \mathcal{L}_i^2$. For an edge $e = (u, v)$ in $\mathcal{G}$, we define $g_D(e) := (g_D(u), g_D(v))$ (see Fig. 10). This operation may generate several edges e between a pair of nodes u', v' of $\mathcal{G}_D$. As in stage B, they are handled differently according to their type. Parallel edges running into the same PAR-input node, that means v' represents a level $\mathcal{L}_i^2$, are reduced to a single edge, whereas two parallel edges e running into an ALT-input node v' are kept. They will be named $(u', v')_0$ and $(u', v')_1$.

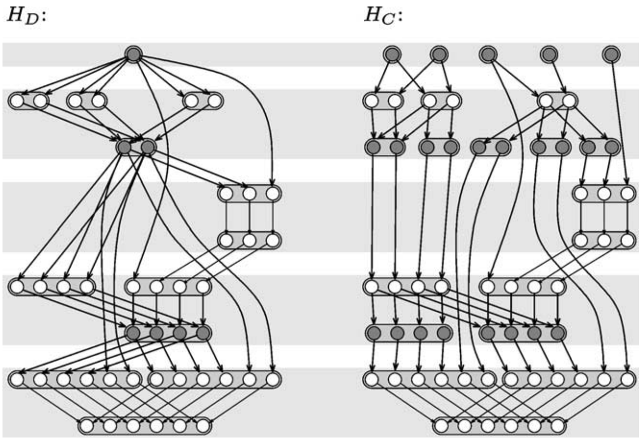

Fig. 12. Transformation from a run H_D to run H_C.

From a run H_D for $\mathcal{G}_D$ one can construct a run for $\mathcal{G}_C$ in a straightforward way. Each execution instance of a node c_i has to be expanded to a cluster of nodes that represent the complete level $\mathcal{L}_i^2$. Connections to execution instances of direct predecessors and successors can then be made in such a way that the overlap does not increase. Simply label all execution instances of duplicates in a consistent way and draw the edge connection correspondingly (see Fig. 12). If $\mathcal{L}_i^2$ has several connections from or to a process u we make sure that in the run for each cluster representing $\mathcal{L}_i^2$ its edges to execution instances of u all choose the same execution instance. Hence, if H_D is k-overlapping so is the run H_C constructed this way.

Lemma 5.6. $\mu(\mathcal{G}_C) = \mu(\mathcal{G}_D)$ *and* $\gamma(\mathcal{G}_C) \leqslant \gamma(\mathcal{G}_D)$.

Again, the inverse transformation $\bar{g}_D$ is local. Furthermore, given a run H_D for $\mathcal{G}_D$ one can construct a run H_C for $\mathcal{G}_C$ for each execution instance and edge independently and in parallel.

5.6. Stage E: reducing the number of ALT*-input nodes*

In the final stage, for each non-empty level $\mathcal{L}_i$ we collapse the number of ALT-input nodes that have the same predecessors to a single node in order to achieve the final property that for all values $i, j, k \leqslant \mu(\mathcal{G})$ with $k = i + j$ and $i \leqslant j$ there exists at most one ALT-input node $v_{i,j}$.

This is done as in the previous transformation. Fig. 10 illustrates the procedures and Fig. 13 shows an example for the corresponding run transformation. Again, the mappings and their inverse are local, and the overlap does not increase.

Lemma 5.7. $\mu(\mathcal{G}_D) = \mu(\mathcal{G}_E)$ *and* $\gamma(\mathcal{G}_D) \leqslant \gamma(\mathcal{G}_E)$.

Summarizing we have shown

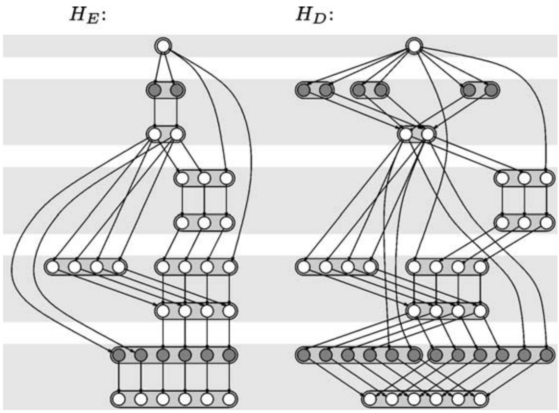

Fig. 13. Transformation from a run H_E to run H_D.

Proposition 5.8. *Every DPG $\mathcal{G}$ can be transformed to a DPG $\mathcal{G}_E$ fulfilling properties (A)–(E) as specified above. In particular, every path in $\mathcal{G}_E$ from its unique source to its unique sink alternates with respect to the input mode of its nodes, $\mu(\mathcal{G}_E) = \mu(\mathcal{G})$, and $\gamma(\mathcal{G}_E) \geqslant \gamma(\mathcal{G})$.*

6. A linear upper bound on the maximal overlap

To finish the proof of Theorem 4.4 we will restrict the class of DPGs still further. Given a graph of type $\mathcal{G}_E$, a corresponding *complete* DPG is obtained from it by expanding its levels $\mathcal{L}_1$ up to $\mathcal{L}_{\mu(\mathcal{G}_E)}$ to their maximal size such that conditions (A) to (E) still hold. This transformation generates a unique DPG that only depends on $\mu(\mathcal{G}_E)$. DPGs in such a normal form turn out to be easier to analyse.

Before discussing the special properties of these graphs we will relate arbitrary DPGs to this normal form. A DPG $\mathcal{G}'$ is a *subgraph* of a DPG $\mathcal{G}$ iff for the underlying graphs the ordinary subgraph relation is fulfilled and the input and output modes of corresponding nodes match. Note that for DPGs this subgraph ordering does not behave monotonically with respect to executability and overlap. For example, consider the graph obtained by adding the two edges (a, i) and (a, j) to the DPG in Fig. 6. This larger graph has a non-overlapping run (to find such a run is not too difficult and we leave it to the reader), whereas the original DPG requires 1-overlap as we have proven.

Thus we need additional restrictions in order to relate $\gamma(\mathcal{G})$ and $\gamma(\mathcal{G}')$ if $\mathcal{G}'$ is a subgraph of $\mathcal{G}$.

Proposition 6.1. *Let $\mathcal{G}'$ be a subgraph of a* PAR*-output DPG $\mathcal{G}$ such that for each node $v \in \mathcal{G}'$ it holds: $|W_{\mathcal{G}'}(v)| = |W_{\mathcal{G}}(v)|$. Then $\gamma(\mathcal{G}') \leqslant \gamma(\mathcal{G})$, in particular if $\mathcal{G}$ is executable the same holds for its subgraph.*

Proof. From the fact that $|W_{\mathcal{G}'}(v)| = |W_{\mathcal{G}}(v)|$ for all nodes $v \in \mathcal{G}'$ we can conclude, that each ALT-input node of $\mathcal{G}'$ has the same predecessors as the corresponding node in $\mathcal{G}$.

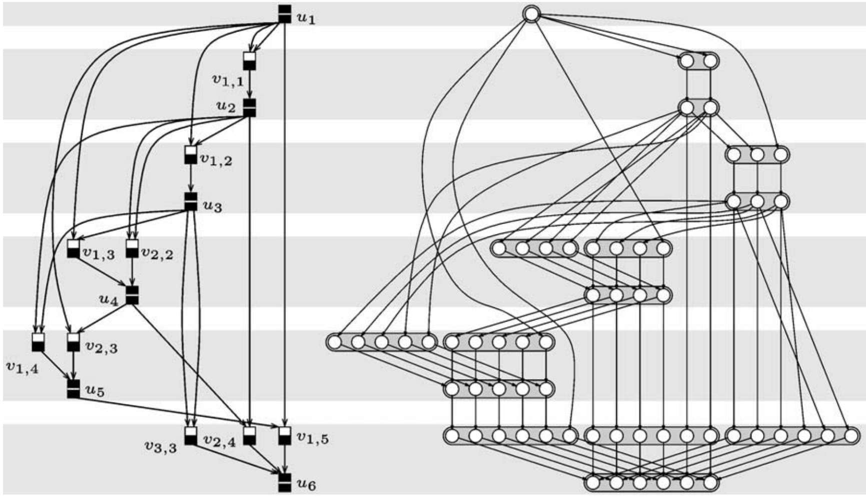

Fig. 14. Left: the complete 6-level DPG $\mathcal{C}_6$; right: a run for $\mathcal{C}_6$.

Hence, a run for $\mathcal{G}'$ can be constructed as a subgraph from the run $H_\mathcal{G}$ such that $W_{\mathcal{G}'}(v) = W_\mathcal{G}(v)$ for all nodes $v \in \mathcal{G}'$. Therefore, the set of predecessors of each execution instance of $t \in W$ in $H_{\mathcal{G}'}$ is a subset of its predecessor execution instances in $H_\mathcal{G}$. □

Definition 6.2. For an integer $\ell > 0$ define a PAR-output DPG $\boldsymbol{\mathcal{C}_\ell} = (V_\ell, E_\ell, I, O)$, called the *complete l-level DPG*, as follows:

$$V_\ell := \{u_1, u_2, \dots, u_\ell\} \cup \{v_{i,j} \mid 1 \leqslant i \leqslant j \text{ and } i + j \leqslant \ell\},$$
$$E_\ell := \big\{(u_i, v_{i,j}), (u_j, v_{i,j}), (v_{i,j}, u_k) \mid 1 \leqslant i \leqslant j,\ 2 \leqslant k \leqslant \ell \text{ and } i + j = k\big\},$$
$$I(u_k) := \mathsf{PAR} \quad \text{and} \quad I(v_{i,j}) := \mathsf{ALT}.$$

It is not hard to check that this graph satisfies all five properties (A)–(E). Moreover, it holds $\mu(\mathcal{C}_\ell) = \ell$. As an example the complete DPG $\mathcal{C}_6$ is illustrated in Fig. 14. As a result of the sequence of transformations described in the previous section $\mathcal{C}_\ell$ is the largest possible graph obtainable when starting from DPGs with μ-value at most ℓ. Thus, by Proposition 6.1 this graph is the worst example that can occur with respect to the maximum overlap. The motivation for considering this special family of graphs is that for these *complete* graphs we are able to prove a general upper bound on the maximum overlap necessary.

Theorem 6.3. $\gamma(\mathcal{C}_\ell) \leqslant \log_2 \ell$.

Before proving this result let us consider its implications in more detail. Given an arbitrary PAR-output DPG $\mathcal{G} = (V, E, I, O)$ with $\ell := \mu(\mathcal{G})$, by the sequence of transformations (A) to (E) we obtain a DPG $\mathcal{G}' = (V', E', I', O')$ with $\mu(\mathcal{G}') = \ell$ and $\gamma(\mathcal{G}) \leqslant \gamma(\mathcal{G}')$. Since $\mathcal{G}'$ satisfies properties (A) to (E) we can consider it being a subgraph of $\mathcal{C}_\ell$. More

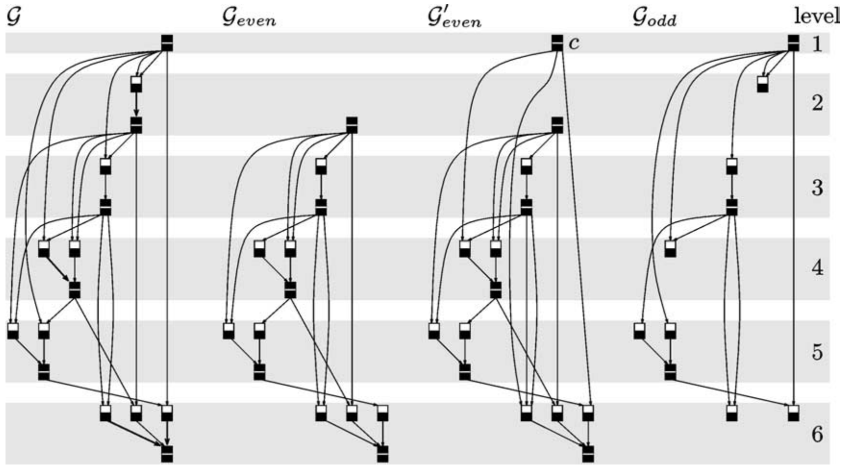

Fig. 15. Splitting the complete PAR-output DPG $\mathcal{G}$ into $\mathcal{G}_{even}$ and $\mathcal{G}_{odd}$. Since $\mathcal{G}_{even}$ is not executable we add one additional node c to get the executable DPG $\mathcal{G}'_{even}$.

precisely, its nodes can be named such that $V' \subseteq V_\ell$, in particular for $u_k \in V'$ it holds $|W(u_k)| = k$ for any $k \leqslant \ell$. Every node of $\mathcal{G}'$ requires the same number of execution instances in $\mathcal{G}'$ as in $\mathcal{C}_\ell$. Hence, Proposition 6.1 implies

Corollary 6.4. $\gamma(\mathcal{G}) \leqslant \gamma(\mathcal{G}') \leqslant \gamma(\mathcal{C}_{\mu(\mathcal{G})})$.

Combining this estimation with Theorem 6.3 we arrive at the claim of Theorem 4.4

$$\gamma(\mathcal{G}) \leqslant \log_2 \mu(\mathcal{G}).$$

Proof of Theorem 6.3. We construct a run H_ℓ for $\mathcal{C}_\ell$ recursively. Consider two induced subgraphs of $\mathcal{C}_\ell$ denoted by $\boldsymbol{\mathcal{G}_{even}}$ and $\boldsymbol{\mathcal{G}_{odd}}$ that are obtained as follows (compare Fig. 15):

$$V_{even} := V_\ell \setminus \{u_1, v_{1,1}\}, \qquad V_{odd} := V_\ell \setminus \{\, u_k, v_{i,j} \mid k, i, j \text{ even}\}.$$

Note that $\mathcal{G}_{odd}$ is executable, while $\mathcal{G}_{even}$ is not if $\ell \geqslant 4$. The process $v_{1,3}$ in $\mathcal{G}_{even}$ requires 1 execution instance, the process $v_{2,2}$ requires 2. But such a situation is in conflict with the PAR-input mode of u_4 which implies that all direct predecessors must have the same number of execution instances. To make $\mathcal{G}_{even}$ executable we add a new source c to the graph and an edge $(c, v_{i,j})$ if i, j are both odd. This new graph—let us call it $\mathcal{G}'_{even}$—possesses a run $H_{even} = (W_{even}, F_{even})$ with the properties

$$\left|W_{even}(v_{i,j})\right| = \lfloor (i+j)/2 \rfloor \quad \forall\, 1 < i, j \text{ with } i + j \leqslant \ell, \tag{6.1}$$

$$\left|W_{even}(u_k)\right| = \lfloor k/2 \rfloor \quad \forall\, 1 \leqslant k \leqslant \ell. \tag{6.2}$$

The construction of H_{even} is straightforward. In particular, it holds

$$\mu(\mathcal{G}'_{even}) = \left|W_{even}(u_\ell)\right| = \lfloor \ell/2 \rfloor.$$

For the recursive step, $\mathcal{G}'_{even}$ is transformed by the function

$$g := g_E \circ g_D \circ g_C \circ g_B \circ g_A.$$

Recall that $\gamma(\mathcal{G}'_{even}) \leqslant \gamma(g(\mathcal{G}'_{even}))$. Moreover, Corollary 6.4 implies $\gamma(g(\mathcal{G}'_{even})) \leqslant \gamma(\mathcal{C}_{\lfloor \ell/2 \rfloor})$. Combining these inequalities we obtain

$$\gamma(\mathcal{G}'_{even}) \leqslant \gamma(\mathcal{C}_{\lfloor \ell/2 \rfloor}).$$

By induction, we can assume $\gamma(\mathcal{C}_{\lfloor \ell/2 \rfloor}) \leqslant \log_2 \lfloor \ell/2 \rfloor$, that means there exists a run $H_{even} = (W_{even}, F_{even})$ for $\mathcal{G}'_{even}$ that is $\log_2 \lfloor \ell/2 \rfloor$-overlapping.

Now consider $\mathcal{G}_{odd}$. For every i, j, with $1 \leqslant i, j$ and $i + j \leqslant \ell$, this graph contains the edge $(v_{i,j}, u_{i+j})$ iff $i + j$ is odd. Therefore, only two cases can occur: either i is odd and j is even, or i is even and j is odd. u_k does not belong to $\mathcal{G}_{odd}$ if k is even. This implies that either $(u_i, v_{i,j}) \in E_{odd}$ and $(u_j, v_{i,j}) \notin E_{odd}$, or $(u_j, v_{i,j}) \in E_{odd}$ and $(u_i, v_{i,j}) \notin E_{odd}$. Therefore, the indegree of a non-sink ALT-input node $v_{i,j}$ in $\mathcal{G}_{odd}$ is 1 and it has only one execution instance. Hence, $\mathcal{G}_{odd}$ is executable and any run of $\mathcal{G}_{odd}$ is non-overlapping. Let $H_{odd} = (W_{odd}, F_{odd})$ denote such a run. We can conclude that

$$\begin{aligned} \left|W_{odd}(u_k)\right| &= \begin{cases} 1 & \text{if } k \text{ is odd,} \\ 0 & \text{if } k \text{ is even,} \end{cases} \\ \left|W_{odd}(v_{i,j})\right| &= \begin{cases} 2 & \text{if } i \text{ and } j \text{ are odd,} \\ 1 & \text{if } i + j \text{ is odd,} \\ 0 & \text{if } i \text{ and } j \text{ are even.} \end{cases} \end{aligned} \tag{6.3}$$

To finish the proof we construct a $\log_2 \ell$-overlapping run $H = (W, F)$ for $\mathcal{C}_\ell$ from H_{even} and H_{odd}. To generate such a run take two copies $H_1 = (W_1, F_1)$ and $H_2 = (W_2, F_2)$ of H_{even} and one copy $H_3 = (W_3, F_3)$ of H_{odd}. Let s_1 denote the execution instance of the additional source c in H_1 and let S_1 be the set containing s_1 and all its direct successors in H_1. Analogously, define s_2 and S_2 for H_2. Below we give the construction for H (see also Fig. 16). Let

$$W := (W_1 \setminus S_1) \cup (W_2 \setminus S_2) \cup W_3.$$

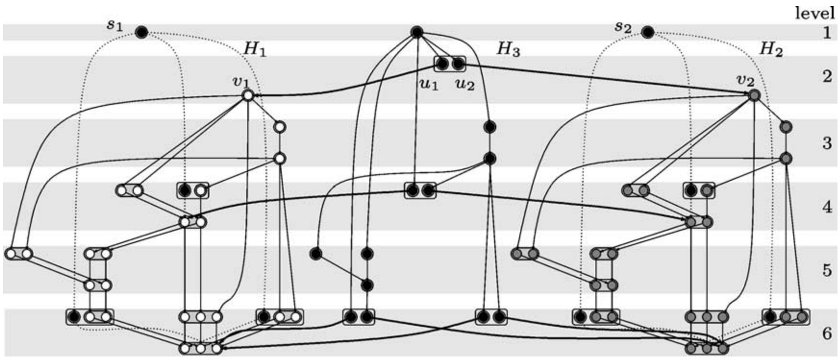

Fig. 16. The construction of a $(\log_2 \ell)$-overlapping run for $\mathcal{C}_\ell$ from H_{odd} and H_{even}.

For any $v \in V_\ell$ we define a set of execution instances $W(v) \subseteq W$ as follows. Let $W(u_1) := W_3(u_1)$, and for $k = 2, 3, \ldots, \ell$ define

$$W(u_k) := \begin{cases} W_1(u_k) \cup W_2(u_k) & \text{if } k \text{ is even,} \\ W_1(u_k) \cup W_2(u_k) \cup W_3(u_k) & \text{if } k \text{ is odd.} \end{cases}$$

For i, j with $1 \leqslant i \leqslant j$ and $i + j = k$ let

$$W(v_{i,j}) := \begin{cases} W_1(v_{i,j}) \cup W_2(v_{i,j}) & \text{if } i \text{ and } j \text{ even,} \\ (W_1(v_{i,j}) \setminus S_1) \cup (W_2(v_{i,j}) \setminus S_2) \cup W_3(v_{i,j}) & \text{if } i \text{ and } j \text{ odd,} \\ W_1(v_{i,j}) \cup W_2(v_{i,j}) \cup W_3(v_{i,j}) & \text{if } i + j \text{ odd.} \end{cases}$$

Since for $v \in V_\ell$ the sets $W_1(v)$, $W_2(v)$, and $W_3(v)$ are disjoint the definition above and Eqs. (6.1)–(6.3) imply

$$\left|W(u_k)\right| = k \quad \text{and} \quad \left|W(v_{i,j})\right| = i + j. \tag{6.4}$$

To construct the edges F of H we proceed as follows: For odd i and j with $1 \leqslant i, j$ and $i + j \leqslant \ell$ let $W_3(v_{i,j}) = \{t_{i,j,1}, t_{i,j,2}\}$, let $z_{i,j,1}$ be a node in $S_1 \cap W_1(v_{i,j})$ and $z_{i,j,2}$ a node in $S_2 \cap W_2(v_{i,j})$. Moreover, let $z_{i+j,1}, z_{i+j,2}$ be direct successors of $z_{i,j,1}$ in H_1, respectively $z_{i,j,2}$ in H_2. Finally, let $\{z_{2,1}\} := W_1(u_2)$ and $\{z_{2,2}\} := W_2(u_2)$. Note that according to our definition of W, both $z_{i,j,1}$ and $z_{i,j,2}$ do not belong to W anymore. Now we choose:

$$\begin{aligned} F := \big((W \times W) \cap (F_1 \cup F_2 \cup F_3)\big) \\ \cup \left\{(t_{i,j,1}, z_{i+j,1}), (t_{i,j,2}, z_{i+j,2}) \mid i, j \text{ odd and } 2 \leqslant i + j \leqslant \ell\right\}. \end{aligned}$$

It is easy to see that H is a run of $\mathcal{C}_\ell$: the equations in (6.4) imply that each node of $\mathcal{C}_\ell$ has the correct number of execution instances. From the definition of the edge set it follows that the connections between the execution instances are correct, too. Moreover, H_1 and H_2 are $\log_2\lfloor \ell/2 \rfloor$-overlapping and H_3 is a non-overlapping run. Let us summarize the essential properties of H:

(1) the graphs H_1, H_2, H_3 are node disjoint,
(2) for every path $v_1 \ldots v_k$ from the source to a sink in $\mathcal{G}_{odd}$ and for every $i \in [1..k-1]$ it holds $|W_3(v_i)| = 1$,
(3) every subrun $\mathcal{R}(t)$ of H consists of either a subgraph of H_1 or of H_2 plus a subgraph of H_3 that includes at most one execution instance of a process.

Therefore, the overlap of H is increased to $1 + \log_2 \ell/2$ at most.

7. Scheduling PAR-output DPGs quickly

Since PAR-output DPGs may specify runs of exponential size, even if the input mode is restricted to ALT (see Fig. 17), it is not clear how to compute a schedule for a given DPG—even a suboptimal one—efficiently simply because the size of the output may be extremely large.

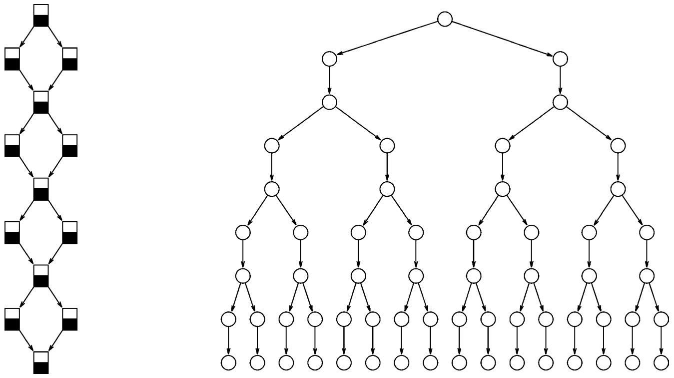

Fig. 17. An ALT-input PAR-output DPG specifying a run of exponential size.

It seems unlikely that the compaction provided by DPGs can always be translated to a similar compaction when describing their corresponding schedules. Thus, even if enough processors are available the question remains how they can synchronize their individual work in order to cooperatively schedule the DPG fast. In this section, we will investigate this problem and show that very little synchronization is actually necessary. A good schedule based on the overlap notion as described in the previous sections can be constructed by a massive-parallel machine in a highly distributive fashion. Since the length of this schedule provides a reasonable approximation for the optimum, the DPG can then be executed efficiently. In parallel, each processor executes the execution instances of the partial schedule it has computed in the construction phase. This can even be done in a quasi on-line fashion intertwining the computation and the execution of each individual partial schedule.

More precisely, we will make the bound of Corollary 4.6 constructive showing that the execution of a program specified by an arbitrary PAR-output DPG can be computed in a distributive fashion such that each subrun is of quadratic size at most. Remember that for DPGs without mode restrictions such runs may not exist (Proposition 3.4).

In the following, we will only consider DPGs with a single sink s and leave the obvious extensions to the multiple sink case to the reader. It has already been observed that the maximum size of a set $W(v)$ of execution instances for a node v, i.e., $\mu(\mathcal{G})$, can be computed sequentially in polynomial time. The method to compute a schedule is based on the reduction described in the previous sections. $\mu(\mathcal{G})$ processors $P_1, \ldots, P_{\mu(\mathcal{G})}$ are used where each processor P_i takes a different execution instance t_i of the sink and computes a corresponding subrun $\mathcal{R}(t_i) = (W_i, F_i)$. These subruns may intersect, but they are constructed in such a way that the composition $(\bigcup_{i=1}^{\mu(\mathcal{G})} W_i, \bigcup_{i=1}^{\mu(\mathcal{G})} F_i)$ generates a run for $\mathcal{G}$. After having determined $\mathcal{R}(t_i)$ the processor P_i computes a single-processor schedule for $\mathcal{R}(t_i)$.

We will now describe a recursive strategy to generate the subruns for $\mathcal{G}$. Each P_i starts with a copy of $\mathcal{G}$. In the first step $\mathcal{G}$ is simplified by the transformations A up to E generating the DPG $\mathcal{G}' := g_E(g_D(g_C(g_B(g_A(\mathcal{G})))))$.

Next, $\mathcal{G}'$ with nodes V' and edges E' is split into an even part $\mathcal{G}_{even}$ and an odd part $\mathcal{G}_{odd}$ as in the previous section. For this split operation we will denote the nodes of $\mathcal{G}'$ by

$v_{i,j}$ and u_k as quoted in property (E). Let u be a new node not in V' with input mode PAR and define the subgraphs by

$$\begin{aligned}
V_{even} &:= V' \cup \{u\} \setminus \{u_1, v_{1,1}\},\\
E_{even} &:= E' \cup \bigl\{(u, v_{i,j}) \mid v_{i,j} \in V' \text{ and } i, j > 1 \text{ are even}\bigr\}\\
&\qquad \setminus \Biggl(\bigl\{(v_{1,1}, u_2)\bigr\} \cup \bigcup_{v_{1,j} \in V'} \bigl\{(u_1, v_{1,j})\bigr\}\Biggr),\\
V_{odd} &:= V' \setminus \{u_k, v_{i,j} \in V' \mid k, i, j \text{ are even}\},\\
E_{odd} &:= E' \cap (V_{odd} \times V_{odd}).
\end{aligned}$$

The size of both graphs is polynomially bounded in the size of $\mathcal{G}$. It holds $\mu(\mathcal{G}_{odd}) \leqslant 2$ and $\mu(\mathcal{G}_{even}) \leqslant \mu(\mathcal{G})/2$. They can be generated in sequential polynomial time.

For the splitting operation we define mappings g_{even}, g_{odd} analogously to the transformations above, and inverses $\bar{g}_{odd}$, $\bar{g}_{even}$. Note that the function $\bar{g}_{odd}$ (respectively $\bar{g}_{even}$) does not cover the graph $\mathcal{G}'$ by its own, whereas $\bar{g}_{odd}(E_{odd})$ and $\bar{g}_{even}(E_{even})$ provide an exact edge cover of $\mathcal{G}'$ when adding the edge $(v_{1,1}, u_2)$.

In order to construct a run for a DPG $\mathcal{G}$ each processor P_i first computes the two split graphs $\mathcal{G}_{even}$ and $\mathcal{G}_{odd}$ and stores the inverse transformations $\bar{g}_A$ to $\bar{g}_E$ and $\bar{g}_{odd}$, $\bar{g}_{even}$. In the recursive step, the first $\ell := \mu(\mathcal{G}_{even})$ processors compute a run $H_1 = (W_1, F_1)$ for the left copy $\mathcal{G}_1$ of $\mathcal{G}_{even}$, and the second ℓ processors a run $H_2 = (W_2, F_2)$ for the right copy $\mathcal{G}_2$ of $\mathcal{G}_{even}$. Furthermore, each processor $P_1, \dots, P_{\mu(\mathcal{G})}$ computes a run $H_3 = (W_3, F_3)$ for $\mathcal{G}_{odd}$ and indicates the two execution instances in $W_3(v_{i,j})$ with odd i and j by $t_{1,i,j}$ and $t_{2,i,j}$. This naming can be done consistently by all processors simultaneously.

By induction we assume that each processor $P_1, \dots, P_\ell$ generates a subrun $\mathcal{R}_1(t_i)$ of a sink of $\mathcal{G}_1$ such that the union of these subruns is a run H_1 for $\mathcal{G}_1$, and similarly each processor $P_{\ell+1}, \dots, P_{2\cdot\ell}$ a subrun $\mathcal{R}_2(t_i)$ of H_2. Now, each of the first ℓ processors replaces the direct successors of execution instances in $W_1(u)$ by the corresponding execution instances $t_{1,i,j}$, and each of the second ℓ processors the direct successors of execution instances in $W_2(u)$ by the corresponding execution instances $t_{2,i,j}$. Furthermore, for $W_1(u_2) = \{t_{1,2}\}$ each processor $P_1, \dots, P_\ell$ adds the edge $(t_{1,1,1}, t_{1,2})$ to its graph, and processors $P_{\ell+1}, \dots, P_{2\cdot\ell}$ the edge $(t_{2,1,1}, t_{2,2})$ if $W_2(u_2) = \{t_{2,2}\}$. Let us denote the resulting graphs by $R_1, \dots, R_{2\cdot\ell}$. The construction of $\mathcal{G}_{even}$ from $\mathcal{G}'$ implies

$$2 \cdot \mu(\mathcal{G}_{even}) \leqslant \mu(\mathcal{G}') \leqslant 2 \cdot \mu(\mathcal{G}_{even}) + 1.$$

If $\mu(\mathcal{G}')$ equals the larger value the last processor $P_{2\cdot\ell+1}$ only computes the run H_3 of $\mathcal{G}_{odd}$. By induction it follows that H_3 contains a subrun for a sink execution instance of $\mathcal{G}'$ in this case. Let $R_{2\cdot\ell+1}$ be this subgraph of H_{odd}, and consider the union

$$H := \Biggl(\bigcup_{1 \leqslant i \leqslant \mu(\mathcal{G}')} W_i, \bigcup_{1 \leqslant i \leqslant \mu(\mathcal{G}')} F_i\Biggr) \quad \text{with } R_i = (W_i, F_i) \text{ for all } 1 \leqslant i \leqslant \mu(\mathcal{G}').$$

From the construction of $\mathcal{G}'$ it follows that its run does not contain multiple edges. Furthermore, the subgraphs R_i are not disjoint, they might have overlapping nodes as well as edges. Analogously to Section 6 it follows that H is a run for $\mathcal{G}'$ with at most $\log_2 \mu(\mathcal{G}')$-overlap.

Assume that for the subruns of $\mathcal{G}_1$ and $\mathcal{G}_2$ and the run of $\mathcal{G}_{odd}$ mappings are given that transform execution instances and edges to the corresponding nodes and edges of the DPGs. Then, one can locally generate the subruns of a run of $\mathcal{G}$ by using the inverse functions $\bar{g}_{odd}, \bar{g}_{even}, \ldots, \bar{g}_A$ step by step such that each processor P_i computes one subrun $\mathcal{R}(t_i)$ for a sink t_i as well as the inverse elements for nodes and edges of this subrun. As in the proof of Theorem 6.3 it follows that the run constructed this way is at most $\log\mu(\mathcal{G})$-overlapping. Since in each recursive step the value of $\mu(\mathcal{G})$ decreases by a factor of two the number of iterations is bounded by $\log\mu(\mathcal{G}) \leqslant |\mathcal{G}|$. Without bothering about architectural details of the parallel machine model—for example, one could choose any standard variant of a parallel random access machine or a network of processors—we can conclude that every processor can perform its computation in polynomial time, hence:

Theorem 7.1. *For any* PAR*-output DPG $\mathcal{G}$, a $\log\mu(\mathcal{G})$-overlapping run can be computed in parallel polynomial time.*

Corollary 7.2. *Given a* PAR*-output DPG $\mathcal{G}$, the program Π represented by $\mathcal{G}$ can be executed by a parallel machine in polynomial time with respect to the size of $\mathcal{G}$.*

Let us finally point out that this result provides a fast parallel approximation to an $\mathcal{NEXPTIME}$-hard problem, namely the problem to compute an optimal schedule for a compactly specified parallel program.

8. Conclusion

Dynamic process graphs are a useful tool to specify parallel programs in a compact way. The compaction ratio, respectively the blow-up when unfolding the specification to an actual run of the program can be exponential. In previous work we have shown that determining the minimum schedule length for DPGs is a $\mathcal{NEXPTIME}$-complete problem, even when we restrict the class of graphs to PAR-output DPGs [4,5].

In this paper we have shown that for PAR-output DPGs the maximum size of a subrun provides an approximation for the optimal schedule length—it can be bounded quadratically in the size of the representation. This improves significantly the best possible general upper bound for the unrestricted class of DPGs which is exponential.

Our technical tool to obtain this bound is a careful analysis of the maximum overlap between different subruns that cannot be avoided. This linear upper bound may not be tight, but a precise analysis of overlap that cannot be avoided seems complicated. We have not even succeeded to design a DPG requiring overlap 2. Furthermore, how can one efficiently recognize DPGs which possess non-overlapping runs and can one find such runs fast? Theorems A and B imply that for certain DPGs the number of different runs can be double exponential. Therefore, a brute-force search will be inefficient.

The upper bound has also been made constructive. For PAR-output DPGs we have described a fast parallel algorithm to construct a low overlapping run and an appropriate schedule. Because of the possibly exponential blowup of runs extensive parallelism may be necessary in order to achieve time efficiency. The parallel algorithm does not require much synchronization between processors, thus these problems can be solved in a highly distributive fashion. For a highly intractable optimization problem this algorithm provides a fast and reasonably precise approximation—a linear approximation ratio compared to the obvious exponential bound.

It remains an open problem to find other *natural* structural properties of DPGs and runs that may yield better approximations for their minimum schedule.

Acknowledgements

We thank the unknown referees for their remarks that helped us to improve the presentation of these results.

References

[1] H. Galperin, A. Wigderson, Succinct representations of graphs, Inform. and Control 56 (1983) 183–198.
[2] S. Ha, E. Lee, Compile-time scheduling and assignment of data-flow program graphs with data-dependent iteration, IEEE Trans. Comput. 40 (1991) 1225–1238.
[3] H. Jung, L. Kirousis, P. Spirakis, Lower bounds and efficient algorithms for multiprocessor scheduling of DAGs with communication delays, Inform. and Comput. 105 (1993) 132–158.
[4] A. Jakoby, M. Liśkiewicz, R. Reischuk, Scheduling dynamic graphs, in: Proc. 16th Symposium on Theoretical Aspects in Computer Science STACS'99, in: Lecture Notes in Comput. Sci., vol. 1563, Springer-Verlag, 1999, pp. 383–392, for a complete version see Electronic Colloquium on Computational Complexity, Report TR01-090, 2001.
[5] A. Jakoby, M. Liśkiewicz, R. Reischuk, The expressive power and complexity of dynamic process graphs, in: Proc. 26th International Workshop on Graph-Theoretic Concepts in Computer Science WG'00, in: Lecture Notes in Comput. Sci., vol. 1928, Springer-Verlag, 2000, pp. 230–242.
[6] T. Lengauer, K. Wagner, The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems, J. CSS 44 (1992) 63–93.
[7] C. Papadimitriou, M. Yannakakis, A note on succinct representations of graphs, Inform. and Control 71 (1986) 181–185.
[8] C. Papadimitriou, M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms, in: Proc. 20th STOC, 1988, pp. 510–513, see also SIAM J. Comput. 19 (1990) 322–328.